Physics-Informed Model-Based Reinforcement Learning

Adithya Ramesh
Robert Bosch Centre for Data Science and Artificial Intelligence
Indian Institute of Technology Madras

Balaraman Ravindran
Robert Bosch Centre for Data Science and Artificial Intelligence
Department of Computer Science and Engineering
Indian Institute of Technology Madras

Abstract

One of the drawbacks of traditional reinforcement learning (RL) algorithms has been their poor sample efficiency. One approach to improve the sample efficiency is model-based RL. In our model-based RL algorithm, we learn a model of the environment, essentially its transition dynamics and reward function, use it to generate imaginary trajectories and backpropagate through them to update the policy, exploiting the differentiability of the model. Intuitively, learning more accurate models should lead to better performance. We focus on robotic systems undergoing rigid body motion without contacts. Recently, there has been growing interest in developing better deep neural network based dynamics models for physical systems, through better inductive biases. We compare two versions of our model-based RL algorithm, one which uses a standard deep neural network based dynamics model and the other which uses a much more accurate, physics-informed neural network based dynamics model. We show that, in model-based RL, model accuracy mainly matters in environments that are sensitive to initial conditions. In these environments, the physics-informed version of our algorithm achieves significantly better average-return and sample efficiency. In environments that are not sensitive to initial conditions, both versions of our algorithm achieve similar average-return, while the physics-informed version achieves better sample efficiency. We measure the sensitivity to initial conditions using the finite-time maximal Lyapunov exponent. We also show that, in challenging environments, where we need a lot of samples to learn, physics-informed model-based RL can achieve better average-return than state-of-the-art model-free RL algorithms such as Soft Actor-Critic, by generating accurate imaginary data.

1 Introduction

Reinforcement learning (RL) can solve sequential decision making problems through trial and error. In recent years, RL has been combined with powerful function approximators such as deep neural networks to solve complex robotics tasks with high-dimensional state and action spaces (Lillicrap et al., 2015; Schulman et al., 2017; Fujimoto et al., 2018; Haarnoja et al., 2018). However, there remain some critical challenges in RL today (Dulac-Arnold et al., 2021). One of them is sample efficiency. Traditional RL algorithms require a lot of interactions with the environment to learn successful policies. In robotics, collecting such large amounts of training data using actual robots is not practical. One solution is to train in simulation. However, for many robotics tasks, developing realistic simulations is hard. Another solution is to improve the sample efficiency. One approach to improve the sample efficiency is model-based RL. Here, we learn a model of the environment, essentially its transition dynamics and reward function, use it to generate imaginary trajectories and use that data to update the policy. This way we need fewer interactions with the actual environment. Some model-based RL algorithms use the model just to generate additional data and update the policy using a model-free algorithm, for e. g., Sutton, 1991; Janner et al., 2019. Other model-based RL algorithms exploit the differentiability of the model and backpropagate through the imaginary trajectories to update the policy, for e. g., Deisenroth and Rasmussen, 2011; Heess et al., 2015; Clavera et al., 2020; Hafner et al., 2019, 2020. We adopt the second approach.

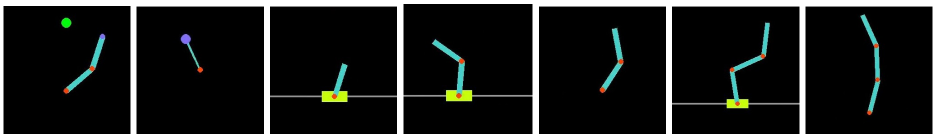
Intuitively, learning more accurate models should lead to better model-based RL performance. Recently there has been growing interest in developing better deep neural network based dynamics models for physical systems through better inductive biases (Lutter et al., 2019a; Greydanus et al., 2019; Lutter et al., 2019b; Zhong et al., 2019, 2020; Cranmer et al., 2020; Finzi et al., 2020; Zhong et al., 2021a). These physics-informed models learn the dynamics much more accurately compared to standard deep neural networks. They also obey the underlying physical laws, such as conservation of energy much better. Previous studies in this area have mostly focused on improving dynamics learning. Some studies have learnt a physics-informed model and used it for downstream tasks such as inverse dynamics control (Lutter et al., 2019a) and energy based control (Lutter et al., 2019b; Zhong et al., 2019). We learn a physics-informed model and use it to train our model-based RL algorithm.

We focus on robotic systems undergoing rigid body motion without contacts. We compare two versions of our model-based RL algorithm, one which uses a standard deep neural network based dynamics model and the other which uses a much more accurate, physics-informed neural network based dynamics model. Our contributions are as follows,

- Our first contribution is combining physics-informed models and model-based RL. We are one of the first to do so.
- Our second contribution is showing that, in model-based RL, model accuracy mainly matters in environments that are sensitive to initial conditions. In these environments, the physics-informed version of our algorithm achieves significantly better average-return and sample efficiency. In environments that are not sensitive to initial conditions, both versions of our algorithm achieve similar average-return, while the physics-informed version achieves better sample efficiency.
- The sensitivity to initial conditions depends on the system dynamics, control policy, degree of actuation and damping. We measure the sensitivity to initial conditions using the finite-time maximal Lyapunov exponent. We compute it using the variational equation (Skokos, 2010), which linearizes the dynamics to estimate how separation vectors evolve with time. The standard variational equation is defined only for autonomous systems and other non-autonomous systems which can be made autonomous. We extend the variational equation to accomodate a control policy. This is our third contribution.
- Our fourth contribution is showing that, in challenging environments, where we need a lot of samples to learn, physics-informed model-based RL can achieve better average-return than state-of-the-art model-free RL algorithms such as Soft Actor-Critic, by generating accurate imaginary data.

2 Environments

We focus on robotic systems undergoing rigid body motion without contacts. The environments considered are shown in Figure 1.



(a) Reacher (b) Pendulum (c) Cartpole (d) Cart-2-pole (e) Acrobot (f) Cart-3-pole (g) Acro-3-bot

Figure 1: We consider robotic systems undergoing rigid body motion without contacts.

We assume that there is no friction. In future work, we plan to include both friction as well as contacts. We provide more details about the environments, such as the task to be accomplished, which joints are acutated, etc, in Appendix A. We develop our own simulations from first principles.

2.1 Lagrangian Mechanics

These systems obey Lagrangian mechanics. Their state consists of generalized coordinates $\mathbf{q}$, which describe the configuration of the system, and generalized velocities $\dot{\mathbf{q}}$, which are the time derivatives of $\mathbf{q}$. The Lagrangian is a scalar quantity defined as $\mathcal{L}(\mathbf{q}, \dot{\mathbf{q}}, t) = \mathcal{T}(\mathbf{q}, \dot{\mathbf{q}}) - \mathcal{V}(\mathbf{q})$, where $\mathcal{T}(\mathbf{q}, \dot{\mathbf{q}})$ is the kinetic energy and $\mathcal{V}(\mathbf{q})$ is the potential energy. Let the motor torques be $\boldsymbol{\tau}$. The Lagrangian equations of motion are given by,

$$\frac{d}{dt}\frac{\partial \mathcal{L}}{\partial \dot{\mathbf{q}}} - \frac{\partial \mathcal{L}}{\partial \mathbf{q}} = \boldsymbol{\tau} \tag{1}$$

For rigid body motion, the kinetic energy is given by $\frac{1}{2}\,\dot{\mathbf{q}}^T\,\mathbf{M}(\mathbf{q})\,\dot{\mathbf{q}}$, where $\mathbf{M}(\mathbf{q})$ is the mass matrix, which is symmetric and positive definite. Hence, the Lagrangian equations of motion become,

$$\mathbf{M}(\mathbf{q})\,\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\,\dot{\mathbf{q}} + \mathbf{G}(\mathbf{q}) = \boldsymbol{\tau} \tag{2}$$

where, $\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\,\dot{\mathbf{q}} = \frac{\partial}{\partial \mathbf{q}}\big(\mathbf{M}(\mathbf{q})\,\dot{\mathbf{q}}\big)\,\dot{\mathbf{q}} - \frac{\partial}{\partial \mathbf{q}}\big(\frac{1}{2}\,\dot{\mathbf{q}}^T\,\mathbf{M}(\mathbf{q})\,\dot{\mathbf{q}}\big)$, is the centripetal / Coriolis term and $\mathbf{G}(\mathbf{q}) = \frac{\partial \mathcal{V}(\mathbf{q})}{\partial \mathbf{q}}$, is the gravitational term.

3 Model-Based RL

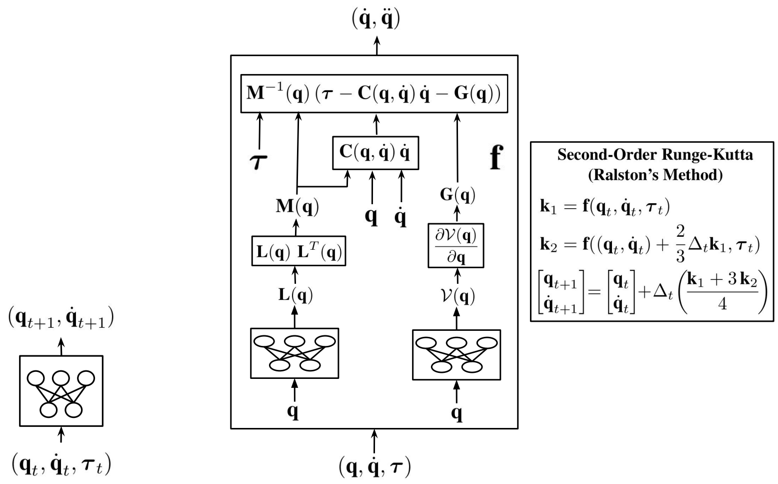
Our model-based RL algorithm essentially iterates over three steps. First is the environment interaction step, where we use the current policy to interact with the environment and gather data. Second is the model learning step, where we use the gathered data to learn the dynamics and reward models. Third is the behaviour learning step, where we use the learned model to generate imaginary trajectories and backpropagate through them to update the policy. We discuss the model learning and behaviour learning steps in detail below.

3.1 Model Learning

In the model learning step, we learn the dynamics and reward models. In dynamics learning, we want to predict the next state, given the current state and action, i. e., we want to learn the transformation $(\mathbf{q}_t, \dot{\mathbf{q}}_t, \boldsymbol{\tau}_t) \rightarrow (\mathbf{q}_{t+1}, \dot{\mathbf{q}}_{t+1})$. The most straightforward solution is to train a standard deep neural network. We refer to this approach as DNN. This is shown in Figure 2(a). Another approach is to utilize the structure of the underlying Lagrangian mechanics. This approach builds upon recent work such as Deep Lagrangian Networks (DeLaN) (Lutter et al., 2019a), DeLaN for energy control (Lutter et al., 2019b) and Lagrangian Neural Networks (Cranmer et al., 2020). We detail this approach here. We use one network to learn the potential energy function $\mathcal{V}(\mathbf{q})$ and another network to learn a lower triangular matrix $\mathbf{L}(\mathbf{q})$, using which we compute the mass matrix as $\mathbf{M}(\mathbf{q}) = \mathbf{L}(\mathbf{q})\,\mathbf{L}^T(\mathbf{q})$. We then compute the centripetal / Coriolis term $\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\,\dot{\mathbf{q}}$ and the gravitational term $\mathbf{G}(\mathbf{q})$. Then, by rearranging Equation 2, we get the acceleration as $\ddot{\mathbf{q}} = \mathbf{M}^{-1}(\mathbf{q})\,(\boldsymbol{\tau} - \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\,\dot{\mathbf{q}} - \mathbf{G}(\mathbf{q}))$. We then numerically integrate the state derivative $(\dot{\mathbf{q}}, \ddot{\mathbf{q}})$ over one time step using second-order Runge-Kutta to compute the next state. We refer to this approach as LNN, short for Lagrangian Neural Network. The entire process is shown in Figure 2(b). In both the DNN and LNN approaches to dynamics learning, we use the L1 error between the predicted state and the ground truth as the loss for training. In reward learning, we want to learn the reward function. In general, the reward is a function of the current state, action and the next state. In our case, the reward only depends on the next state. Hence, we train a network to map the next state to the reward. We use the L1 error between the predicted reward and the ground truth as the loss for training.

3.2 Behaviour Learning

In the behaviour learning step, we use the learned model to generate imaginary trajectories and backpropagate through them to update the policy. We build upon the Dreamer algorithm (Hafner et al., 2019, 2020). We adopt an actor-critic approach. The critic aims to predict the expected discounted


$(\mathbf{q}_{t+1}, \dot{\mathbf{q}}_{t+1})$
$(\mathbf{q}_t, \dot{\mathbf{q}}_t, \boldsymbol{\tau}_t)$
$(\dot{\mathbf{q}}, \ddot{\mathbf{q}})$
$\mathbf{M}^{-1}(\mathbf{q})\,(\boldsymbol{\tau} - \mathbf{C}(\mathbf{q},\dot{\mathbf{q}})\,\dot{\mathbf{q}} - \mathbf{G}(\mathbf{q}))$
$\boldsymbol{\tau}$
$\mathbf{C}(\mathbf{q},\dot{\mathbf{q}})\,\dot{\mathbf{q}}$
$\mathbf{f}$
$\mathbf{M}(\mathbf{q})$
$\mathbf{q}$ $\dot{\mathbf{q}}$
$\mathbf{G}(\mathbf{q})$
$\frac{\partial \mathcal{V}(\mathbf{q})}{\partial \mathbf{q}}$
$\mathbf{L}(\mathbf{q})\ \mathbf{L}^T(\mathbf{q})$
$\mathbf{L}(\mathbf{q})$
$\mathcal{V}(\mathbf{q})$
$\mathbf{q}$
$\mathbf{q}$
$(\mathbf{q}, \dot{\mathbf{q}}, \boldsymbol{\tau})$
Second-Order Runge-Kutta (Ralston's Method)
$\mathbf{k}_1 = \mathbf{f}(\mathbf{q}_t, \dot{\mathbf{q}}_t, \boldsymbol{\tau}_t)$
$\mathbf{k}_2 = \mathbf{f}((\mathbf{q}_t, \dot{\mathbf{q}}_t) + \frac{2}{3}\Delta_t \mathbf{k}_1, \boldsymbol{\tau}_t)$
$\begin{bmatrix}\mathbf{q}_{t+1}\\ \dot{\mathbf{q}}_{t+1}\end{bmatrix} = \begin{bmatrix}\mathbf{q}_t\\ \dot{\mathbf{q}}_t\end{bmatrix} + \Delta_t \left(\frac{\mathbf{k}_1 + 3\,\mathbf{k}_2}{4}\right)$

(a) DNN

(b) LNN

Figure 2: Two approaches to dynamics learning : In DNN, we use a standard deep neural network. In LNN, we utilize the structure of the underlying Lagrangian mechanics.

return from a given state. We train the critic to regress the λ-return (Schulman et al., 2015). We stabilize the critic training by computing the λ-return using a target network that is updated every 100 critic updates. The critic loss function is given by $L(w) = \mathbb{E}\,[\sum_{t=0}^{T-1} \frac{1}{2}(\,V(s_t; w) - \text{sg}\,(V'_\lambda(s_t))\,)^2]$, where,

$$V'_\lambda(s_t) = \begin{cases} r_t + \gamma\,(\,(1-\lambda)V'(s_{t+1}; w') + \lambda V'_\lambda(s_{t+1})\,) & \text{if } t < T \\ V'(s_t; w') & \text{if } t = T \end{cases} \tag{3}$$

We stop the gradients around the target (denoted by the sg(.) function), as is typical in the literature. We use a stochastic actor. The actor aims to output actions that lead to states that maximize the expected discounted return. We train the actor to maximize the same λ-return that was computed to train the critic. We add an entropy term to the actor objective to encourage exploration. The overall actor loss function is given by $L(\theta) = -\,\mathbb{E}\,[\sum_{t=0}^{T-1}[\,V'_\lambda(s_t) - \eta \log \pi(a_t|s_t;\theta)\,]]$. To backpropagate through sampled actions, we use the reparameterization trick (Kingma and Welling, 2013). The actor network outputs the mean μ and standard deviation σ of a Gaussian distribution, from which we obtain the action as $a_t = \tanh(\mu_\theta(s_t) + \sigma_\theta(s_t) \cdot \epsilon)$, where $\epsilon \sim \mathcal{N}(0, \mathbb{I})$. We summarize our overall model-based RL algorithm in Algorithm 1.

```
Initialize networks with random weights.
Execute random actions for $K$ episodes to initialize replay buffer.
for each episode do
    // Model Learning
    for $N_1$ times do
        Draw mini batch of transitions from replay buffer. Fit dynamics and reward models.
    end
    // Behaviour Learning
    for $N_2$ times do
        Draw mini batch of states from replay buffer. From each state, imagine a trajectory of length $T$.
        Actor Loss $L(\theta) = -\,\mathbb{E}\,[\sum_{t=0}^{T-1}[\,V'_\lambda(s_t) - \eta \log \pi(a_t|s_t;\theta)\,]]$
        Critic Loss $L(w) = \mathbb{E}\,[\sum_{t=0}^{T-1} \frac{1}{2}(\,V(s_t; w) - \text{sg}\,(V'_\lambda(s_t))\,)^2]$
        Backpropagate gradients through imaginary trajectories to update actor, critic.
        Every 100 updates, copy critic weights into critic target.
    end
    // Environment Interaction
    Interact with environment for one episode using current policy. Add transitions to replay buffer.
end
```

Algorithm 1: Model-Based RL Algorithm.

3.3 Experiments

We train two versions of our model-based RL algorithm, one which uses the DNN approach for dynamics learning and the other which uses the LNN approach. We refer to them as MBRL-DNN and MBRL-LNN respectively. In both versions, we use an imagination horizon of 16 time steps. In addition, we train a state-of-the-art model-free RL algorithm, Soft Actor-Critic (SAC) (Haarnoja et al., 2018), to serve as a baseline. We train each algorithm on five random seeds.

4 Results and Analysis

We record the results from the experiments in Table 1. The training curves are shown in Figure 3. We summarize the results below,

- Across environments, MBRL-LNN achieves significantly lower dynamics error than MBRL-DNN. The reward error for both methods are similar.
- In Reacher, Pendulum and Cartpole, all the methods, i. e., MBRL-DNN, MBRL-LNN and SAC, successfully solve the task and achieve similar average-return.
- In Cart-2-pole, again, all the methods successfully solve the task. However, in terms of average-return, MBRL-LNN $>$ SAC $>$ MBRL-DNN.
- In Acrobot, Cart-3-pole and Acro-3-bot, only MBRL-LNN and SAC successfully solve the task, while MBRL-DNN is unsuccessful. Again, in terms of average-return, MBRL-LNN $>$ SAC $>$ MBRL-DNN.
- Across environments, MBRL-LNN achieves similar or better average return than MBRL-DNN and SAC, while requiring fewer samples. In other words, MBRL-LNN achieves better sample efficiency.

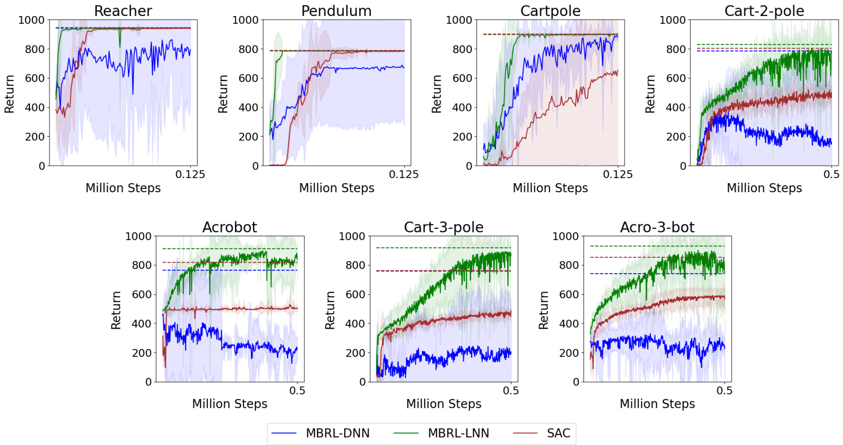


Figure 3: Training curves for MBRL-DNN, MBRL-LNN and SAC experiments. The solid curves represent the mean performance over all the seeds. The shaded region represents the 95 % confidence interval. The dashed lines indicate the best average-return after convergence.

We try to understand the results better. Reacher, Pendulum and Cartpole appear to be simple environments where all methods perform well. Cart-2-pole, Acrobot, Cart-3-pole and Acro-3-bot appear to be more challenging environments, where there is a noticeable difference in the performance of different methods. We try to understand what makes these latter environments more challenging and why certain methods perform better in these environments than others.

4.1 Environment Dynamics

As a first step, we try to characterize the underlying dynamics of each environment by calculating their Lyapunov exponents. Lyapunov exponents measure the rate of separation of trajectories which

Environment	Method	Steps	Dynamics Error	Reward Error	Average Return	Solved (Y/N)	Free Fall MLE	Finite Time MLE	Trajectory Error
Reacher	MBRL-DNN	0.25 M	126.57e-5	7.2217e-4	942.7	**Y**	0.0247	-0.0051	1.0984
	MBRL-LNN	**0.125 M**	**4.7557e-5**	**8.4078e-4**	**943.4**	**Y**			**0.2036**
	SAC	1.25 M	–	–	943.2	**Y**			–
Pendulum	MBRL-DNN	0.25 M	108.93e-5	**8.198e-4**	**787.3**	**Y**	0.0346	-0.0811	0.6933
	MBRL-LNN	**0.125 M**	**3.4559e-5**	8.3962e-4	786.7	**Y**			**0.1099**
	SAC	1.25 M	–	–	786.8	**Y**			–
Cartpole	MBRL-DNN	0.25 M	149.31e-5	**7.5226e-4**	899.4	**Y**	0.0718	-0.4626	0.5141
	MBRL-LNN	**0.125 M**	**6.014e-5**	8.6215e-4	**900.0**	**Y**			**0.1645**
	SAC	1.25 M	–	–	897.4	**Y**			–
Cart-2-pole	MBRL-DNN	2 M	499.94e-5	16.049e-4	786.4	**Y**	1.1330	0.3241	2.6164
	MBRL-LNN	**0.5 M**	**70.389e-5**	**8.8135e-4**	**828.3**	**Y**			**0.9744**
	SAC	10 M	–	–	801.7	**Y**			–
Acrobot	MBRL-DNN	2 M	799.90e-5	8.848e-4	764.5	N	1.0340	0.4265	4.8975
	MBRL-LNN	**0.5 M**	**48.046e-5**	**7.9032e-4**	**911.8**	**Y**			**2.2121**
	SAC	10 M	–	–	817.8	**Y**			–
Cart-3-pole	MBRL-DNN	2 M	1196.0e-5	17.044e-4	757.0	N	3.0904	0.8099	18.2503
	MBRL-LNN	**0.5 M**	**61.3e-5**	**6.5141e-4**	**916.6**	**Y**			**1.3601**
	SAC	10 M	–	–	759.4	**Y**			–
Acro-3-bot	MBRL-DNN	2 M	1322.0e-5	16.6256e-4	739.7	N	3.2265	0.6230	16.0338
	MBRL-LNN	**0.5 M**	**156.34e-5**	**10.308e-4**	**929.4**	**Y**			**2.8153**
	SAC	10 M	–	–	853.5	**Y**			–

Table 1: Overall results from MBRL-DNN, MBRL-LNN and SAC experiments.

start from nearby initial states. If the initial separation vector is $\mathbf{u}_0$ and the separation vector at time t is $\mathbf{u}_t$, then the Lyapunov exponent is defined as $\lambda = \frac{1}{t}\log\frac{\|\mathbf{u}_t\|}{\|\mathbf{u}_0\|}$. The rate of separation varies based on the orientation of the initial separation vector. In general, there is a spectrum of Lyapunov exponents, equal in number to the state space dimension. It is common to refer to the largest among them as the maximal Lyapunov exponent (MLE). The rate of separation is maximum along the direction associated with the MLE. An arbitrary initial separation vector will typically contain some component in this direction, and because of the exponential growth rate, this component will dominate the evolution. The MLE is a measure of the system's sensitivity to initial conditions. It also represents how quickly numerical errors will accumulate. Thus, it is also a measure of the predictability.

Finite-Time MLE : We are interested in knowing how fast nearby trajectories will diverge, under a trained RL policy, over finite-time. Hence, we compute the finite-time MLE. We consider the period from the start of an episode, till the agent reaches the goal state, which typically takes several hundred time steps. The simplest method to compute the finite-time MLE is to evolve two trajectories starting from nearby initial states. A more efficient and reliable method is to use the variational equation (Skokos, 2010), which linearizes the dynamics to estimate how separation vectors evolve with time. The standard variational equation is defined only for autonomous systems, $\dot{\mathbf{s}} = \mathbf{f}(\mathbf{s})$, and non-autonomous systems of the form $\dot{\mathbf{s}} = \mathbf{f}(\mathbf{s}, t)$, which can be made autonomous by treating the time t as an additional dependent variable with the trivial evolution equation $\dot{t} = 1$. The variational equation is given by $\dot{\mathbf{u}} = \frac{d\mathbf{f}}{d\mathbf{s}}\,\mathbf{u}$. We extend the variational equation to accomodate a control policy. Let the dynamics in the presence of a control policy be described by $\dot{\mathbf{s}} = \mathbf{f}(\mathbf{s}, \mathbf{a})$, where $\mathbf{a} = \boldsymbol{\pi}(\mathbf{s})$. The extended variational equation is given by,

$$\dot{\mathbf{u}} = \frac{d\mathbf{f}}{d\mathbf{s}}\,\mathbf{u} = (\frac{\partial\mathbf{f}}{\partial\mathbf{s}} + \frac{\partial\mathbf{f}}{\partial\mathbf{a}}\,\frac{d\mathbf{a}}{d\mathbf{s}})\,\mathbf{u} \tag{4}$$

We consider a random initial separation vector of unit length. We then jointly integrate the system dynamics along with the extended variational equation to compute how the separation vector evolves with time. We renormalize the separation vector periodically and each time, we use the Lyapunov exponent definition to compute a fresh estimate of the finite-time MLE and then update its running average. We continue this process until the agent reaches the goal state. We compute the finite-time MLE for policies of MBRL-DNN, MBRL-LNN and SAC, and average the results. We record the results in Table 1. We find that Reacher, Pendulum and Cartpole have a relatively small, negative, finite-time MLE. This implies that they are less sensitive to initial conditions and more predictable. Whereas, Cart-2-pole, Acrobot, Cart-3-pole and Acro-3-bot have a relatively large, positive, finite-time MLE. This implies that they are more sensitive to initial conditions and less predictable.

Free Fall MLE : To get an idea of the system's inherent sensitivity to initial conditions, independent of any control policy, we compute the long-term MLE under free fall. We call this the free fall MLE. We follow the same procedure we used to compute the finite-time MLE, with some minor changes. One change is that, we use the standard variational equation. The other change is that, we continue the MLE computation process for a sufficiently long time, until the running average converges. We record the results in Table 1. We find that Reacher, Pendulum and Cartpole have a relatively small, positive, free fall MLE, implying low inherent sensitivity to initial conditions. Whereas, Cart-2-pole, Acrobot, Cart-3-pole and Acro-3-bot have a relatively large, positive, free fall MLE, implying high inherent sensitivity to initial conditions.

4.2 Actuation

In Appendix A, we describe which joints are acutated in each environment. Reacher and Pendulum are fully-actuated. The rest of the environments are underactuated. The level of underactuation varies. Cartpole and Cart-2-pole are somewhat less underactuated, while Acrobot, Cart-3-pole and Acro-3-bot are somewhat more underactuated. Underactuation by itself makes it hard to learn successful policies. Underactuation combined with no damping leads to somewhat unrestrained motion.

4.3 Imaginary Trajectories

Next, we assess the quality of the imaginary trajectories produced by MBRL-DNN and MBRL-LNN. We generate 10,000 imaginary trajectories of length 16 time steps and also generate the corresponding ground truth trajectories. First, we assess the accuracy of the imaginary trajectories. For each imaginary trajectory, we compute the state error at each time step, i. e., the L1 error between the predicted state and the true state. We plot the average state error as a function of time in Figure 4. For each imaginary trajectory, we also compute the trajectory error, which is simply the sum of the state errors along the trajectory. We record the average trajectory error in Table 1. We find that, MBRL-LNN has a low trajectory error across environments. MBRL-DNN's trajectory error varies. In Reacher, Pendulum, Cartpole and Cart-2-pole, it has a moderate trajectory error. In Acrobot, Cart-3-pole and Acro-3-bot, it has a high trajectory error. Across environments, MBRL-LNN has lower trajectory error than MBRL-DNN, i. e, MBRL-LNN's imaginary trajectories are more accurate.

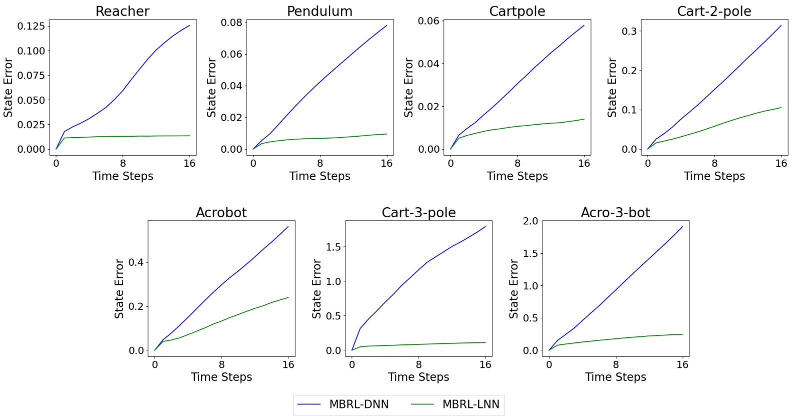


Figure 4: Accuracy of imaginary trajectories : State Error vs Time.

Next, we assess how well the imaginary trajectories respect the underlying physical laws, such as conservation of energy. In our environments, the work done by external non-conservative forces, i. e, the actuators, is non-zero. Hence, the total energy of the system will change. The change in total energy must equal the work done by external non-conservative forces, for energy to be conserved. We define the energy error as the absolute difference between the two quantities. For each imaginary trajectory, we compute the energy error at each time step. We plot the average energy error as a function of time in Figure 5. We find that, across environments, MBRL-LNN has lower energy error than MBRL-DNN, i. e, MBRL-LNN's imaginary trajectories conserve energy better.

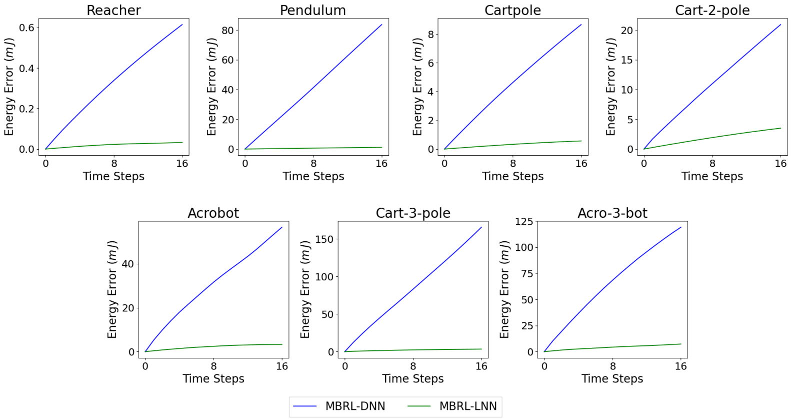


Figure 5: Conservation of energy in imaginary trajectories : Energy Error vs Time.

4.4 Discussion

Cart-2-pole, Acrobot, Cart-3-pole and Acro-3-bot are underactuated and have high inherent sensitivity to initial conditions, which is characterized by a large free fall MLE. Underactuation by itself makes it hard to learn successful policies. Underactuation combined with no damping leads to somewhat unrestrained motion. Such unrestrained motion in a system with a large free fall MLE leads to sensitivity to initial conditions, which is characterized by a large finite-time MLE. Sensitivity to initial conditions leads to high variance in the agent's actual as well as imaginary trajectories. It also leads to poor predictability, which affects critic learning, as it is concerned with predicting the expected discounted return, and hence also affects actor learning, as our methods follow an actor-critic approach. Under these conditions, we need a lot of samples to learn.

SAC is successful in all four of these environments. However, it is affected by the issue of a lot of samples being needed to learn. MBRL-LNN overcomes this issue by generating accurate imaginary data. This is the main reason why SAC achieves lower average-return than MBRL-LNN in these environments.

Sensitivity to initial conditions makes it challenging for standard deep neural networks to learn accurate dynamics models. In Acrobot, Cart-3-pole and Acro-3-bot, MBRL-DNN has a large dynamics error. As numerical errors accumulate fast in these environments, its imaginary trajectories have high trajectory error. Hence, it is unable to learn good policies and is unsuccessful in these environments. Cart-2-pole is a borderline case. Here, MBRL-DNN achieves a moderate dynamics error. Hence, even though numerical errors accumulate fast, its imaginary trajectories only have moderate trajectory error. Hence, it is able to learn a reasonably good policy and is successful.

In all four of these environments, MBRL-LNN achieves a small dynamics error due to its physics-based inductive biases. Hence, even though numerical errors accumulate fast, its imaginary trajectories have low trajectory error. Hence, it is able to learn good policies and is successful in these environments. Thus, MBRL-LNN achieves better average-return and sample efficiency than both MBRL-DNN and SAC in these environments.

In contrast, Reacher, Pendulum and Cartpole are fully-actuated and/or have low inherent sensitivity to initial conditions, which is characterized by a small free fall MLE. Hence, they are less sensitive to initial conditions, which is characterized by a small finite-time MLE. In these environments, MBRL-

LNN has a low dynamics error, MBRL-DNN has a moderate dynamics error and numerical errors accumulate slowly. Hence, MBRL-LNN has a low trajectory error, MBRL-DNN has a moderate trajectory error and both methods are able to learn good policies and are successful, and so is the case with SAC. In these environments, MBRL-LNN achieves similar average-return to MBRL-DNN and SAC, but achieves better sample efficiency.

5 Conclusion

We apply model-based RL to robotic systems undergoing rigid body motion without contacts. In our model-based RL algorithm, we learn a model of the environment, essentially its transition dynamics and reward function, use it to generate imaginary trajectories and backpropagate through them to update the policy, exploiting the differentiability of the model. We compare two versions of our algorithm, one which uses a standard deep neural network based dynamics model and the other which uses a much more accurate, physics-informed neural network based dynamics model.

We show that, in model-based RL, model accuracy mainly matters in environments that are sensitive to initial conditions. In these environments, the physics-informed version of our algorithm achieves significantly better average-return and sample efficiency. In environments that are not sensitive to initial conditions, both versions of our algorithm achieve similar average-return, while the physics-informed version achieves better sample efficiency.

The sensitivity to initial conditions depends on the system dynamics, control policy, degree of actuation and damping. We measure the sensitivity to initial conditions using the finite-time maximal Lyapunov exponent. We compute it using the variational equation, which linearizes the dynamics to estimate how separation vectors evolve with time. The standard variational equation is defined only for autonomous systems and other non-autonomous systems which can be made autonomous. We extend the variational equation to accomodate a control policy.

We also show that, in challenging environments, where we need a lot of samples to learn, physics-informed model-based RL can achieve better average-return than state-of-the-art model-free RL algorithms such as Soft Actor-Critic, by generating accurate imaginary data.

6 Future Work

In future work, we plan to consider friction, as it is present in most real world robots. As far as learning friction, there are two approaches. In the black box approach (Zhong et al., 2020), the friction is learnt directly from data using a neural network. In the white box approach (Lutter et al., 2019b), an analytic friction model is used and its parameters alone are learnt from data. We plan to pursue the white box approach.

In future work, we also plan to consider contact dynamics, as it is present in many robotics tasks like manipulation and locomotion. There are predominantly two approaches to simulate contact dynamics, the linear complementarity approach (LCP) (Stewart and Trinkle, 1996; Anitescu and Potra, 1997) and the convex optimization approach (Todorov et al., 2012). LCP was the standard approach for a long time. Convex optimization has similar accuracy and is more efficient. We plan to adopt the latter approach and use differentiable convex optimization layers (Agrawal et al., 2019) to learn differentiable contact models which can extend LNNs. There has been some recent work in this direction by Zhong et al., 2021b. However they assume that the robot's environment is known and use this knowledge to compute the active contacts and their jacobians. We plan to do away with this assumption by equipping the robot with sensors to perceive its environment.

Upon learning a physics-informed model that captures rigid body dynamics, friction and contact dynamics, we plan to train a model-based RL algorithm.

Appendix

A Environment Details

The task to be accomplished in each environment is as follows,

- Reacher : Control a fully-actuated two-link manipulator in the horizontal plane to reach a fixed target location.
- Pendulum : Swing up and balance a simple pendulum.
- Cartpole : Swing up and balance an unactuated pole by applying forces to a cart at its base.
- Cart-2-pole : One extra pole is added to Cartpole. Again, only the cart is actuated.
- Acrobot : Control a two-link manipulator to swing up and balance. Only the second pole is actuated.
- Cart-3-pole : One extra pole is added to Cart-2-pole. Only the cart and the third pole are actuated.
- Acro-3-bot : One extra pole is added to Acrobot. Only the first and the third poles are actuated.

In all the environments, there are no terminal states. Each episode consists of 1000 time steps. The total reward over an episode is in the range $[0, 1000]$.

B Implementation Details

Table 2 and Table 3 list the hyperparameters and network architectures used in our model-based RL experiments. For SAC, we use the same hyperparameters and network architectures as the original SAC paper (Haarnoja et al., 2018).

Parameter	Value
Random episodes at start of training (K)	10
Replay buffer size	10^5
Batch size for model learning	64
Model learning batches per episode (N_1)	10^4
Batch size for behaviour learning	64
Behaviour learning batches per episode (N_2)	10^3
Imagination horizon (T)	16
Discount factor (γ)	0.99
Lambda (λ)	0.95
Entropy weightage (η)	10^{-4}
Gradient clipping norm	100
Optimizer	AdamW
Learning rate	3×10^{-4}

Table 2: Model-Based RL Hyperparameters.

Network	Architecture
Critic	$256 , 256 , 1$
Actor	$256 , 256 , 2 \times \dim(a)$
DNN	$64 , 64 , \dim(s)$
LNN L	$64 , 64 , \frac{\dim(s) \times (\dim(s)+2)}{8}$
LNN V	$64 , 64 , 1$
Reward	$64 , 64 , 1$

Table 3: Model-Based RL Network Architectures.

C Long-Horizon Imaginary Trajectories

Although during behaviour learning we use an imagination horizon of only 16 time steps, in Figures 6 and 7, we plot imaginary trajectories of horizon 100 time steps, to see how far into the future our models can predict accurately. We find that, across environments, MBRL-LNN is able to accurately predict quite far into the future, while MBRL-DNN is unable to.

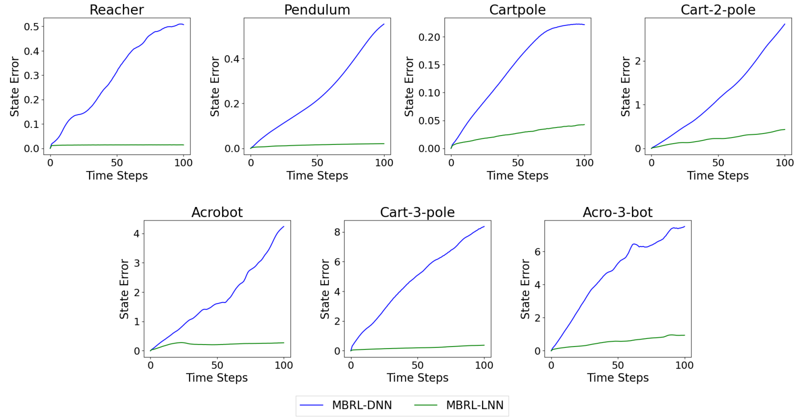


Figure 6: Accuracy of long-horizon imaginary trajectories : State Error vs Time.

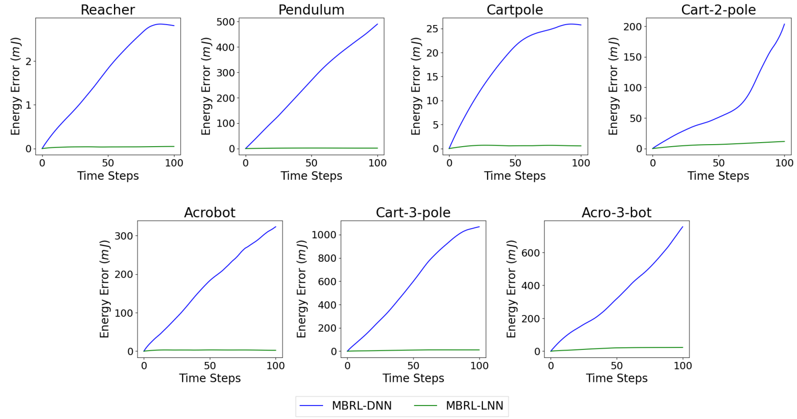


Figure 7: Conservation of energy in long-horizon imaginary trajectories : Energy Error vs Time.

References

Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International conference on machine learning*, pages 1587–1596. PMLR, 2018.

Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, et al. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018.

Gabriel Dulac-Arnold, Nir Levine, Daniel J Mankowitz, Jerry Li, Cosmin Paduraru, Sven Gowal, and Todd Hester. Challenges of real-world reinforcement learning: definitions, benchmarks and analysis. *Machine Learning*, 110(9):2419–2468, 2021.

Richard S Sutton. Dyna, an integrated architecture for learning, planning, and reacting. *ACM Sigart Bulletin*, 2(4):160–163, 1991.

Michael Janner, Justin Fu, Marvin Zhang, and Sergey Levine. When to trust your model: Model-based policy optimization. *Advances in Neural Information Processing Systems*, 32, 2019.

Marc Deisenroth and Carl E Rasmussen. Pilco: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on machine learning (ICML-11)*, pages 465–472. Citeseer, 2011.

Nicolas Heess, Gregory Wayne, David Silver, Timothy Lillicrap, Tom Erez, and Yuval Tassa. Learning continuous control policies by stochastic value gradients. *Advances in neural information processing systems*, 28, 2015.

Ignasi Clavera, Violet Fu, and Pieter Abbeel. Model-augmented actor-critic: Backpropagating through paths. *arXiv preprint arXiv:2005.08068*, 2020.

Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination. *arXiv preprint arXiv:1912.01603*, 2019.

Danijar Hafner, Timothy Lillicrap, Mohammad Norouzi, and Jimmy Ba. Mastering atari with discrete world models. *arXiv preprint arXiv:2010.02193*, 2020.

M. Lutter, C. Ritter, and J. Peters. Deep lagrangian networks: Using physics as model prior for deep learning. In *International Conference on Learning Representations (ICLR)*, 2019a.

Samuel Greydanus, Misko Dzamba, and Jason Yosinski. Hamiltonian neural networks. *Advances in neural information processing systems*, 32, 2019.

Michael Lutter, Kim Listmann, and Jan Peters. Deep lagrangian networks for end-to-end learning of energy-based control for under-actuated systems. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 7718–7725. IEEE, 2019b.

Yaofeng Desmond Zhong, Biswadip Dey, and Amit Chakraborty. Symplectic ode-net: Learning hamiltonian dynamics with control. In *International Conference on Learning Representations*, 2019.

Yaofeng Desmond Zhong, Biswadip Dey, and Amit Chakraborty. Dissipative symoden: Encoding hamiltonian dynamics with dissipation and control into deep learning. In *ICLR 2020 Workshop on Integration of Deep Neural Models and Differential Equations*, 2020.

Miles Cranmer, Sam Greydanus, Stephan Hoyer, Peter Battaglia, David Spergel, and Shirley Ho. Lagrangian neural networks. In *ICLR 2020 Workshop on Integration of Deep Neural Models and Differential Equations*, 2020.

Marc Finzi, Ke Alexander Wang, and Andrew G Wilson. Simplifying hamiltonian and lagrangian neural networks via explicit constraints. *Advances in neural information processing systems*, 33: 13880–13889, 2020.

Yaofeng Desmond Zhong, Biswadip Dey, and Amit Chakraborty. Benchmarking energy-conserving neural networks for learning dynamics from data. In *Learning for Dynamics and Control*, pages 1218–1229. PMLR, 2021a.

Ch Skokos. The lyapunov characteristic exponents and their computation. In *Dynamics of Small Solar System Bodies and Exoplanets*, pages 63–135. Springer, 2010.

John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

David E Stewart and Jeffrey C Trinkle. An implicit time-stepping scheme for rigid body dynamics with inelastic collisions and coulomb friction. *International Journal for Numerical Methods in Engineering*, 39(15):2673–2691, 1996.

Mihai Anitescu and Florian A Potra. Formulating dynamic multi-rigid-body contact problems with friction as solvable linear complementarity problems. *Nonlinear Dynamics*, 14(3):231–247, 1997.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ international conference on intelligent robots and systems*, pages 5026–5033. IEEE, 2012.

Akshay Agrawal, Brandon Amos, Shane Barratt, Stephen Boyd, Steven Diamond, and J Zico Kolter. Differentiable convex optimization layers. *Advances in neural information processing systems*, 32, 2019.

Yaofeng Desmond Zhong, Biswadip Dey, and Amit Chakraborty. Extending lagrangian and hamiltonian neural networks with differentiable contact models. *Advances in Neural Information Processing Systems*, 34:21910–21922, 2021b.